LEARN C++
Skill up with our free tutorials

2.4 — Introduction to function parameters and arguments

ALEX[1] **MARCH 30, 2024**

In the previous lesson, we learned that we could have a function return a value back to the function's caller. We used that to create a modular *getValueFromUser* function that we used in this program:

```cpp
#include <iostream>

int getValueFromUser()
{
    std::cout << "Enter an integer: ";
    int input{};
    std::cin >> input;

    return input;
}

int main()
{
    int num { getValueFromUser() };

    std::cout << num << " doubled is: " << num * 2 << '\n';

    return 0;
}
```

However, what if we wanted to put the output line into its own function as well? You might try something like this:

```cpp
#include <iostream>

int getValueFromUser()
{
    std::cout << "Enter an integer: ";
    int input{};
    std::cin >> input;

    return input;
}

// This function won't compile
void printDouble()
{
    std::cout << num << " doubled is: " << num * 2 << '\n';
}

int main()
{
    int num { getValueFromUser() };

    printDouble();

    return 0;
}
```

This won't compile, because function *printDouble* doesn't know what identifier *num* is. You might try defining num as a variable inside function printDouble():

```cpp
void printDouble()
{
    int num{}; // we added this line
    std::cout << num << " doubled is: " << num * 2 << '\n';
}
```

While this addresses the compiler error and makes the program compile-able, the program still doesn't work correctly (it always prints "0 doubled is: 0"). The core of the problem here is that function *printDouble* doesn't have a way to access the value the user entered.

We need some way to pass the value of variable *num* to function *printDouble* so that *printDouble* can use that value in the function body.

Function parameters and arguments

In many cases, it is useful to be able to pass information *to* a function being called, so that the function has data to work with. For example, if we wanted to write a function to add two numbers, we need some way to tell the function which two numbers to add when we call it. Otherwise, how would the function know what to add? We do that via function parameters and arguments.

A **function parameter** is a variable used in the header of a function. Function parameters work almost identically to variables defined inside the function, but with one difference: they are initialized with a value provided by the caller of the function.

Function parameters are defined in the function header by placing them in between the parenthesis after the function name, with multiple parameters being separated by commas.

Here are some examples of functions with different numbers of parameters:

```cpp
// This function takes no parameters
// It does not rely on the caller for anything
void doPrint()
{
    std::cout << "In doPrint()\n";
}

// This function takes one integer parameter named x
// The caller will supply the value of x
void printValue(int x)
{
    std::cout << x  << '\n';
}

// This function has two integer parameters, one named x, and one named y
// The caller will supply the value of both x and y
int add(int x, int y)
{
    return x + y;
}
```

An **argument** is a value that is passed *from* the caller *to* the function when a function call is made:

```cpp
doPrint(); // this call has no arguments
printValue(6); // 6 is the argument passed to function printValue()
add(2, 3); // 2 and 3 are the arguments passed to function add()
```

Note that multiple arguments are also separated by commas.

How parameters and arguments work together

When a function is called, all of the parameters of the function are created as variables, and the value of each of the arguments is *copied* into the matching parameter (using copy initialization). This process is called **pass by value**. Function parameters that utilize pass by value are called **value parameters**.

For example:

```
#include <iostream>

// This function has two integer parameters, one named x, and one named y
// The values of x and y are passed in by the caller
void printValues(int x, int y)
{
    std::cout << x << '\n';
    std::cout << y << '\n';
}

int main()
{
    printValues(6, 7); // This function call has two arguments, 6 and 7

    return 0;
}
```

When function *printValues* is called with arguments *6* and *7*, *printValues*'s parameter *x* is created and initialized with the value of *6*, and *printValues*'s parameter *y* is created and initialized with the value of *7*.

This results in the output:

```
6
7
```

Note that the number of arguments must generally match the number of function parameters, or the compiler will throw an error. The argument passed to a function can be any valid expression (as the argument is essentially just an initializer for the parameter, and initializers can be any valid expression).

Fixing our challenge program

We now have the tool we need to fix the program we presented at the top of the lesson:

```cpp
#include <iostream>

int getValueFromUser()
{
    std::cout << "Enter an integer: ";
    int input{};
    std::cin >> input;

    return input;
}

void printDouble(int value) // This function now has an integer parameter
{
    std::cout << value << " doubled is: " << value * 2 << '\n';
}

int main()
{
    int num { getValueFromUser() };

    printDouble(num);

    return 0;
}
```

In this program, variable *num* is first initialized with the value entered by the user. Then, function *printDouble* is called, and the value of argument *num* is copied into the *value* parameter of function *printDouble*. Function *printDouble* then uses the value of parameter *value*.

Using return values as arguments

In the above problem, we can see that variable *num* is only used once, to transport the return value of function *getValueFromUser* to the argument of the call to function *printDouble*.

We can simplify the above example slightly as follows:

```cpp
#include <iostream>

int getValueFromUser()
{
    std::cout << "Enter an integer: ";
    int input{};
    std::cin >> input;

    return input;
}

void printDouble(int value)
{
    std::cout << value << " doubled is: " << value * 2 << '\n';
}

int main()
{
    printDouble(getValueFromUser());

    return 0;
}
```

Now, we're using the return value of function *getValueFromUser* directly as an argument to function *printDouble*!

Although this program is more concise (and makes it clear that the value read by the user will be used for nothing else), you may also find this "compact syntax" a bit hard to read. If you're more comfortable sticking with the version that uses the variable instead, that's fine.

How parameters and return values work together

By using both parameters and a return value, we can create functions that take data as input, do some calculation with it, and return the value to the caller.

Here is an example of a very simple function that adds two numbers together and returns the result to the caller:

```
#include <iostream>

// add() takes two integers as parameters, and returns the result of their
sum
// The values of x and y are determined by the function that calls add()
int add(int x, int y)
{
    return x + y;
}

// main takes no parameters
int main()
{
    std::cout << add(4, 5) << '\n'; // Arguments 4 and 5 are passed to
function add()
    return 0;
}
```

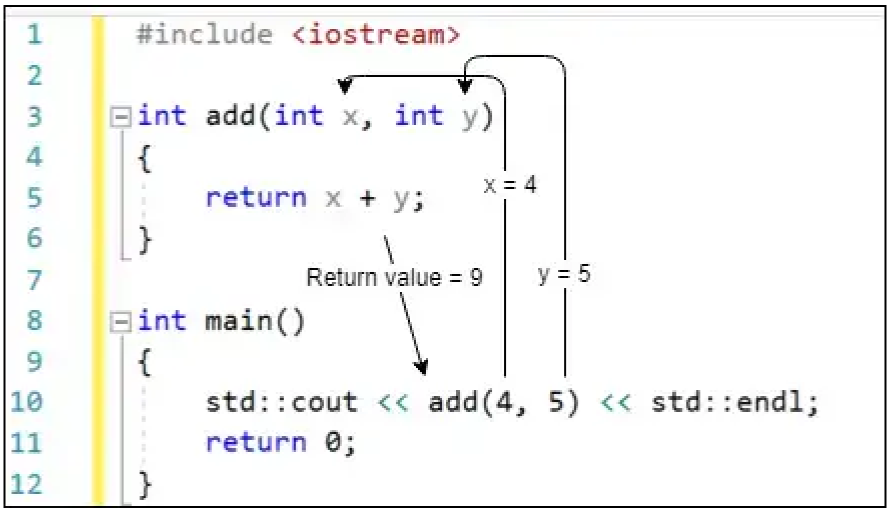
Execution starts at the top of *main*. When `add(4, 5)` is evaluated, function *add* is called, with parameter *x* being initialized with value *4*, and parameter *y* being initialized with value *5*.

The *return statement* in function *add* evaluates *x + y* to produce the value *9*, which is then returned back to *main*. This value of *9* is then sent to *std::cout* to be printed on the console.

Output:

```
9
```

In pictorial format:



```cpp
#include <iostream>

int add(int x, int y)
{
    return x + y;
}

int main()
{
    std::cout << add(4, 5) << std::endl;
    return 0;
}
```

More examples

Let’s take a look at some more function calls:

```cpp
#include <iostream>

int add(int x, int y)
{
    return x + y;
}

int multiply(int z, int w)
{
    return z * w;
}

int main()
{
    std::cout << add(4, 5) << '\n'; // within add() x=4, y=5, so x+y=9
    std::cout << add(1 + 2, 3 * 4) << '\n'; // within add() x=3, y=12, so
x+y=15

    int a{ 5 };
    std::cout << add(a, a) << '\n'; // evaluates (5 + 5)

    std::cout << add(1, multiply(2, 3)) << '\n'; // evaluates 1 + (2 * 3)
    std::cout << add(1, add(2, 3)) << '\n'; // evaluates 1 + (2 + 3)

    return 0;
}
```

This program produces the output:

```
9
15
10
7
6
```

The first statement is straightforward.

In the second statement, the arguments are expressions that get evaluated before being passed. In this case, *1 + 2* evaluates to *3*, so *3* is copied to parameter *x*. *3 * 4* evaluates to *12*, so *12* is copied to parameter *y*. *add(3, 12)* resolves to *15*.

The next pair of statements is relatively easy as well:

```
int a{ 5 };
std::cout << add(a, a) << '\n'; // evaluates (5 + 5)
```

In this case, *add()* is called where the value of *a* is copied into both parameters *x* and *y*. Since *a* has value *5*, *add(a, a)* = *add(5, 5)*, which resolves to value *10*.

Let's take a look at the first tricky statement in the bunch:

```
std::cout << add(1, multiply(2, 3)) << '\n'; // evaluates 1 + (2 * 3)
```

When the function *add* is executed, the program needs to determine what the values for parameters *x* and *y* are. *x* is simple since we just passed it the integer *1*. To get a value for parameter *y*, it needs to evaluate *multiply(2, 3)* first. The program calls *multiply* and initializes *z* = *2* and *w* = *3*, so *multiply(2, 3)* returns the integer value *6*. That return value of *6* can now be used to initialize the *y* parameter of the *add* function. *add(1, 6)* returns the integer *7*, which is then passed to std::cout for printing.

Put less verbosely:
add(1, multiply(2, 3)) evaluates to *add(1, 6)* evaluates to *7*

The following statement looks tricky because one of the arguments given to *add* is another call to *add*.

```
std::cout << add(1, add(2, 3)) << '\n'; // evaluates 1 + (2 + 3)
```

But this case works exactly the same as the prior case. add(2, 3) resolves first, resulting in the return value of *5*. Now it can resolve add(1, 5), which evaluates to the value *6*, which is passed to std::cout for printing.

Less verbosely:
add(1, add(2, 3)) evaluates to *add(1, 5)* => evaluates to *6*

Unreferenced parameters

In certain cases, you will encounter functions that have parameters that are not used in the body of the function. These are called **unreferenced parameters**.

This can happen when a function parameter was once used, but is not used any longer.

As a trivial example:

```cpp
void doSomething(int count) // warning: unreferenced parameter count
{
    // This function used to do something with count but it is not used any
longer
}

int main()
{
    doSomething(4);

    return 0;
}
```

Just like with unused local variables, your compiler will probably warn that variable `count` has been defined but not used.

Key insight

If the unused function parameter were simply removed, then any existing call to the function would break (because the function call would be supplying more arguments than the function could accept).

In a function definition, the name of a function parameter is optional. Therefore, in cases where a function parameter needs to exist but is not used in the body of the function, you can simply omit the name. A parameter without a name is called an **unnamed parameter**:

```cpp
void doSomething(int) // ok: unnamed parameter will not generate warning
{
}
```

The Google C++ style guide recommends using a comment to document what the unnamed parameter was:

```cpp
void doSomething(int /*count*/)
{
}
```

For advanced readers

There are other cases in advanced C++ where unreferenced function parameters occur. For example, C++ uses the existence of an unreferenced function parameter to differentiate between an overload of prefix operator++ (e.g. `++foo`) and postfix operator++ (e.g. `foo++`). We cover this in lesson 21.8 -- Overloading the increment and decrement operators (https://www.learncpp.com/cpp-tutorial/overloading-the-increment-and-decrement-operators/)[2].

Author's note

If unnamed parameters don't make sense to you yet, don't worry. We'll encounter them again in future lessons, when we have more context to explain when they are useful.

Best practice

When a function parameter exists but is not used in the body of the function, do not give it a name. You can optionally put a name inside a comment.

Conclusion

Function parameters and return values are the key mechanisms by which functions can be written in a reusable way, as it allows us to write functions that can perform tasks and return retrieved or calculated results back to the caller without knowing what the specific inputs or outputs are ahead of time.

Quiz time

Question #1

What's wrong with this program fragment?

```cpp
#include <iostream>

void multiply(int x, int y)
{
    return x * y;
}

int main()
{
    std::cout << multiply(4, 5) << '\n';

    return 0;
}
```

Hide Solution (javascript:void(0))[3]

> multiply() has a return type of void, meaning it is a non-value returning function. Since the function is trying to return a value (via a return statement), this function will produce a compiler error. The return type should be int.

Question #2

What two things are wrong with this program fragment?

```cpp
#include <iostream>

int multiply(int x, int y)
{
    int product{ x * y };
}

int main()
{
    std::cout << multiply(4) << '\n';

    return 0;
}
```

Hide Solution (javascript:void(0))[3]

> Problem 1: main() passes one argument to multiply(), but multiply() requires two arguments.
> Problem 2: multiply() doesn't have a return statement.

Question #3

What value does the following program print?

```cpp
#include <iostream>

int add(int x, int y, int z)
{
    return x + y + z;
}

int multiply(int x, int y)
{
    return x * y;
}

int main()
{
    std::cout << multiply(add(1, 2, 3), 4) << '\n';

    return 0;
}
```

Show Solution (javascript:void(0))[3]

Question #4

Write a function called doubleNumber() that takes one integer parameter. The function should return double the value of the parameter.

Hide Solution (javascript:void(0))[3]

```
int doubleNumber(int x)
{
    return 2 * x;
}
```

Question #5

5. Write a complete program that reads an integer from the user, doubles it using the doubleNumber() function you wrote in the previous quiz question, and then prints the doubled value out to the console.

Hide Solution (javascript:void(0))[3]

```
#include <iostream>

int doubleNumber(int x)
{
    return 2 * x;
}

int main()
{
    int x{};
    std::cin >> x;
    std::cout << doubleNumber(x) << '\n';

    return 0;
}
```

Note: You may come up with other (similar) solutions. There are often many ways to do the same thing in C++.

Next lesson
2.5 Introduction to local scope

4

Back to table of contents

5

Previous lesson
2.3 Void functions (non-value returning functions)

6

7

B U URL INLINE CODE C++ CODE BLOCK HELP!

Leave a comment...

Name*

Email*

Notify me about replies:

POST COMMENT

Find a mistake? Leave a comment above!

Avatars from https://gravatar.com/[9] are connected to your provided email address.

990 COMMENTS Newest

Kenneth Onwubuya
April 17, 2024 4:29 pm

```
#include <iostream>

using namespace std;

int getuserData()  //Function for getting user data
{
    cout << "Enter a Number: ";
    int z{};
    cin >> z;

    return z;

}

int doubleNumber(int x) // function that does the doubling
{

    return x*2;
}

void printDoubleNumber(int y) // function to print the value
{
    cout << "this is the double: " << y <<'\n';
}


int main()
{
   int num {getuserData()};
   printDoubleNumber(doubleNumber(num));


    return 0;
}
```

0 Reply

sajLMAO
April 16, 2024 8:35 am

#include <iostream>

int userValue()
{
std::cout << " enter an integer: ";

int input{};
std::cin >> input;

return input;
}

void doubleValue(int value)
{
std::cout << value << " Double the number is : " << value * 2 << '\n';

}

int main() {

doubleValue(userValue());

return 0;
}

0 Reply

:DDDD
April 15, 2024 8:40 am

My straight forward solution

```
#include <iostream>

int doubleNumber(int x)
{
    return x * 2;
}

int main()
{
    std::cout << "Enter an integer: ";
    int uInput;
    std::cin >> uInput;
    std::cout << doubleNumber(uInput) << "\n";
    return 0;
}
```

0 Reply

TastiestBlueberry
April 12, 2024 1:17 pm

is this solution right or did i overthink the problem to Question #5 ? haha

```
#include <iostream>

int doubleNumber(int x)
{
    int calcResult{ x * 2 };

    return calcResult;
}

int main()
{
    std::cout << "Enter an integer to double: ";

    int x{};

    std::cin >> x;

    std::cout << "Your number times 2 is: " << doubleNumber(x);

    return 0;
}
```

0 Reply

Copernicus
April 8, 2024 2:10 am

Question 5:

```
#include <iostream>

int getNumberFromUser()
{
    std::cout << "Enter a number: ";
    int number = { 0 };
    std::cin >> number;

    return number;
}

int multipleNumber(int number, int multipler)
{
    return number * multipler;
}

void displayNumber(int number)
{
    std::cout << number << '\n';
}

int main()
{
    displayNumber(multipleNumber(getNumberFromUser(), 2));

    return 0;
}
```

0 Reply

Jeb
April 6, 2024 10:55 pm

```
#include <iostream>

int askForNumber()
{
    //ask user to imput number
    std::cout << "Input an integer: ";

    //init variable then save the input number to it
    int x{};
    std::cin >> x;

    return x;
}

int doubleNumber(int value)
{
    return value * 2;
}

void printNumber(int numToPrint)
{
    std::cout << "This number doubled is: " << numToPrint;
}

int main()
{
    //ask the user for a number
    int num{ askForNumber() };

    //double and then print the number
    printNumber(doubleNumber(num));

    return 0;
}
```

Last edited 11 days ago by Jeb

0 Reply

Linus
April 5, 2024 7:36 pm

```cpp
#include <iostream>

int GetValueFromUser();
int DoubleNum(int num);
int main()
{
    std::cout << DoubleNum(GetValueFromUser());
    return EXIT_SUCCESS;
}
int DoubleNum(int num) {
    return num * 2;
}
int GetValueFromUser() {
    std::cout << "Type int ";
    int input;
    std::cin >> input;
    return input;
}
```

1 Reply

Vin
April 2, 2024 10:47 pm

why do i feel like im just complicating things xd

```cpp
#include <iostream>

//Doubling numbers
int doubleNum(int x)
{
    return 2 * x;
}

//Tripling numbers
int tripleNum(int y)
{
    return 3 * y;
}

//Get values from users
int usersValue()
{
    int num{};
    std::cout << "Enter an integer: ";
    std::cin >> num;

    return num;
}

//printing the double numbers
void printDouble()
{
    std::cout << "Double that number is: " << doubleNum(usersValue()) <<
'\n' << '\n';
}

//printing the triple numbers
void printTriple()
{
    std::cout << "Triple that number is: " << tripleNum(usersValue()) <<
'\n' << '\n';
}

int main()
{
    std::cout << "===DOUBLING NUMBERS===" << '\n';
    printDouble();

    std::cout << "===TRIPLING NUMBERS===" << '\n';
    printTriple();

    return 0;
}
```

0 Reply

Linus

Reply to Vin 10 April 10, 2024 12:38 pm

it may feel like it because, making a function for every possibility is not the best

0 Reply

Kojo Bailey

Reply to Vin 10 April 4, 2024 6:05 am

The formatting's messed up, since it says `Double that number is: Enter an integer:`

Easy fix though, if you initialise a `buffer` variable first (name doesn't matter)

```cpp
#include <iostream>

/** Double an integer. */
int double_int(int x)
{
    return x * 2;
}

/** Triple an integer. */
int triple_int(int x)
{
    return x * 3;
}

/** Get integer from user and return. */
int get_user_int()
{
    int num{};
    std::cout << "Enter an integer: ";
    std::cin >> num;
    std::cout << "\n";

    return num;
}

/** Get integer from user and output it doubled. */
void print_double()
{
    int buffer{ get_user_int() };
    std::cout << "Double that number is: " << double_int(buffer) <<
'\n';
}

/** Get integer from user and output it tripled. */
void print_triple()
{
    int buffer{ get_user_int() };
    std::cout << "Triple that number is: " << triple_int(buffer) <<
'\n';
}

int main()
{
    std::cout << "=== DOUBLING NUMBERS ===" << '\n';
    print_double();

    std::cout << '\n';

    std::cout << "=== TRIPLING NUMBERS ===" << '\n';
    print_triple();

    return 0;
}
```

0 Reply

Brian
March 29, 2024 11:37 am

I dont understand unnamed parameters. Why would I define a function with a unnamed parameter?

0 Reply

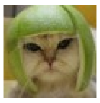
Kojo Bailey

Reply to Brian 11 April 4, 2024 6:16 am

Here's an example using a **forward declaration**, where you declare a function first and define it later:

```
#include <iostream>

int add(int, int);

int main() {
    std::cout << add(2, 3);
}

int add(int a, int b) {
    return a + b;
}
```

1 Reply



Alex Author

Reply to Brian 11 March 30, 2024 12:59 pm

In most cases, you wouldn't. But there are a few exceptions. If the explanation in this section doesn't make sense to you yet, just stay patient -- there are examples in future lessons that will provide more context around cases where this is useful.

2 Reply

Newchallenger147

March 16, 2024 4:36 am

Any feedback? did I solve this pretty good?

```
#include <iostream>

int getDouble() {

    std::cout << "Enter an integer: ";
    int input{};
    std::cin >> input;

    return input;
}


int multiply(int x)
{
    return x * 2;
}


int main()
{

    int num{ getDouble() };


    std::cout << "double that value is: " << multiply(num);

    return 0;

}
```

0 Reply

Pinyan Chen

Reply to Newchallenger147 [12] March 17, 2024 8:29 pm

I don't know where the problem you said.
I think it's OK.

```
#include <iostream>
int get() {
    int input{};
    std::cin >> input;
    return input;
}
int mul(int x) {
    return x * 2;
}
int main()
{
    int num{ get() };
    std::cout << mul(num) << std::endl;
}
```

0 Reply

Thingol
March 13, 2024 9:43 am

Hey, can someone explain to me why when printing two return values inputted by an user from functions used as arguments by another function shows them inconsistent as for what gets executed first. Here's the code:

```
#include <iostream>
int getXFromUser()
{
    std::cout << "Enter your first number: ";
    int inputX{};
    std::cin >> inputX;

    return inputX;

}
int getYFromUser()
{
    std::cout << "Enter your second number: ";
    int inputY{};
    std::cin >> inputY;

    return inputY;

}
void printMulti(int x, int y)
{
    std::cout << x << " multiplyed by " << y << " is " << x * y << '\n';
}
int main()
{

    printMulti( getXFromUser(), getYFromUser());

    return 0;

}
```

`printMulti( getXFromUser(), getYFromUser());` ,even though getXFromUser() is on the left, it gets executed after getYFromUser(). Switching their positions works, but really want to learn what causes this.

Insanely helpful site to learn c++, cheers!

1 Reply

Alex Author
Reply to Thingol 13 March 13, 2024 10:44 pm

https://www.learncpp.com/cpp-tutorial/operator-precedence-and-associativity/#unspecified

1 Reply

ClynnJ
March 5, 2024 2:50 pm

```cpp
#include <iostream>

int getValue(){
    std::cout << "Enter a value: ";
    int num{};
    std::cin >> num;

    return num;
}

int doubleValue(int x){
    return 2 * x;
}

int main(){
    std::cout << doubleValue(getValue()) << " value now doubled!!!\n";

    return 0;
}
```

1 Reply

AssClappicus
March 2, 2024 11:45 am

Bros making me do math on the last question

1 Reply

concat
February 28, 2024 6:48 pm

```cpp
#include <iostream>

int doublenum(int x) {
    return x * 2;
}
int getvalue() {
    std::cout << "enter an int: ";
    int input{};
    std::cin >> input;

    return input;
}

int main() {
    std::cout << doublenum(getvalue()) << '\n';
}
```

Last edited 1 month ago by concat

1 Reply

Tudor
Reply to concat 14 February 29, 2024 11:22 am

```
#include <iostream>

int doubledNumber(int x)
{
    return x * 2;
}

int inputUser()
{
    int num{ 0 };
    std::cout << "Enter an integer: ";
    std::cin >> num;

    return num;

}

int main()
{
    int value{ inputUser() };
    std::cout << value << " doubled is: " << doubledNumber(value);

    return 0;
}
```

1 Reply

Anonymous
February 28, 2024 1:31 pm

```
#include <iostream>

int doubleNumber(int x) {
    return x * 2;
}

int main() {
    int integer;

    std::cout << "Enter a number: ";
    std::cin >> integer;
    std::cout << doubleNumber(integer) << "\n";

    return 0;
}
```

Last edited 1 month ago by Anonymous

0 Reply

Asmo
February 27, 2024 8:48 am

```
#include <iostream>

int takeInput()
{
    std::cout << "enter a integer: " ;
    int a{} ;
    std::cin >> a ;
    return a ;
}

void print(int num)
{
    std::cout << "doubled no is " << num*2 << '\n' ;
}

int main()
{
    print(takeInput()) ;
    return 0 ;
}
```

2 Reply

Appreciate
February 21, 2024 3:52 am

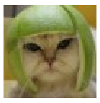
Why do I feel that the program becomes complicated when I split the code into multiple fragment to DRY my code?

```cpp
#include <iostream>

int enterValue()
{
    int x{};
    std::cin >> x;

    return x;
}

int multiNum(int x, int y)
{
    return x * y;
}

int subNum(int x, int y)
{
    return x - y;
}

void printNum()
{
    int x{ enterValue() };
    int y{ enterValue() };

    std::cout << x << "  *  " << y << "  = " << multiNum(x, y) << '\n';
    std::cout << x << "  -  " << y << "  = " << subNum(x, y) << '\n';
}

int main()
{
    std::cout << "\nPlease enter 2 numbers separated by space  :  ";

    printNum();

    return 0;
}
```

Please enter 2 number separated by space : 4 6
4 * 6 = 24
4 - 6 = -2

Last edited 1 month ago by Appreciate

0 Reply



Alex Author

Reply to Appreciate 15 February 23, 2024 1:08 pm

It is a bit more work to break everything into reusable pieces. But it pays off as programs get more complex.

4 Reply

IceFloe
February 19, 2024 7:02 pm

It may be cumbersome, but I had such a solution:

Question 5:

```
#include <iostream>

int doubleNumber (int x)
{
        return 2 * x;
}

void userNumber ()
{
        int user {};
        std::cout << "Enter an integer: ";

        std::cin >> user;

        int doubleUser {doubleNumber (user)};             std::cout << 
"Double is: " << doubleUser << '\n';

}

int main ()                                 {
        userNumber();
        return 0;
}
```

But there was also the first solution, which turned out to be wrong:

```
#include <iostream>

int doubleNumber (int x)
{
        return 2 * x;
}

void userNumber (int user)
{
        std::cout << "Enter an integer: ";
        std::cin >> user;
}

int main ()
{
        int user2 {userNumber()};
        std::cout << " Double variable: " << doubleNumber (user2);
        return 0;
}
```

I will be glad if you point out what I did wrong)))

Last edited 1 month ago by IceFloe

0 Reply

Alex Author
Reply to IceFloe 16 February 19, 2024 9:24 pm

Your latter program is closer to correct.

In the latter program, `user` should be a local variable, not a function parameter, as we don't expect the caller to pass in value for `user`.

`userNumber()` also should return the value to the caller.

0 Reply

llumi
February 16, 2024 12:02 pm

why does this work? i was playing around with the first code and find out if u removed the curly brackets when defining num and it would somehow work and display the inputted number doubled correctly:

```cpp
#include <iostream>

int getValueFromUser()
{
    std::cout << "Enter an integer: ";
    int input{};
    std::cin >> input;

    return input;
}

void printDouble()
{
    int num;
    std::cout << num << " doubled is: " << num * 2 << '\n';
}

int main()
{
    int num { getValueFromUser() };

    printDouble();

    return 0;
}
```

Last edited 2 months ago by llumi

0 Reply

Alex Author
Reply to Ilumi 17 February 17, 2024 6:06 pm

In `printDouble()`, you are using `num` without initializing it. This is undefined behavior, and the result could be anything.

0 Reply

jojy
February 14, 2024 2:54 pm

bro is literally a god by making this FREE LIKE WHAT THANK YOU BRO

0 Reply

Soren
February 7, 2024 9:36 pm

So, I'm having a hard time describing the purpose of an Unreferenced Parameter. Are they simply the byproduct of editing and tweaking code? Why would one write a function that has a parameter that is unused in the body of the function?

0 Reply

Alex Author
Reply to Soren 18 February 8, 2024 10:16 am

Most often they're parameters that were used and are used no longer. However, there are other cases where this can occur. One case is when overloading operator++, which has two versions that are differentiated by the presence or absence of an unreferenced parameter. There are other cases where this kind of thing can occur organically. We cover some of these in future lessons.

1 Reply

Chayim
January 30, 2024 5:09 am

For question 5 I wrote it like this, is it correct?

```cpp
#include <iostream>

int getNumber()
{
    std::cout << "Enter Your Age: ";
    int age{};
    std::cin >> age;

 return age;
}

int doubleNumber(int x)
{
   std::cout << "Your Age Doubled is: ";
    return  x * 2;
}

int main()
{
    std::cout << doubleNumber(getNumber());

  return 0;
}
```

Last edited 2 months ago by Chayim

0 Reply

Patrick
Reply to Chayim 19 January 31, 2024 11:47 am

This will work but it could be improved. In the doubleNumber function you print the line "Your Age Doubled is: " which will give the correct results for this problem, but what if we need to use this function somewhere else in our code? In this case I would recommend adding this print statement into main, because that is where we are concerned specifically with this use of the function and simply return x * 2 within the doubleNumber function.

1 Reply

Aaron
January 24, 2024 7:52 am

In this code, are values x and y defined on lines 7 and 8 of function printValues?

#include <iostream>

// This function has two integer parameters, one named x, and one named y
// The values of x and y are passed in by the caller
void printValues(int x, int y)
{
std::cout << x << '\n';

```
std::cout << y << '\n';
}

int main()
{
printValues(6, 7); // This function call has two arguments, 6 and 7

return 0;
}
```

0 Reply

Patrick
Reply to Aaron 20 January 31, 2024 11:49 am

The variables are defined in the function header "void printValues(int x, int y)" this defines two parameter values which are passed in within the main function and used solely within the scope of the printValues function.

0 Reply

alb
January 24, 2024 4:51 am

```
#include <iostream>

int doubleNumber(int x)
{
return x * 2;
}

int main()
{
// Include doubleNumber() function
int doubleNumber(int x);

// Tell user what the program does
std::cout << "This program will double the number inserted. " << '\n';
std::cout << " " << '\n';

// Ask user for input
std::cout << "Enter a number: " << '\n';
std::cout << " " << '\n';

// Wait for user input
int x{};
std::cin >> x;
```

// Display result of user input into program
std::cout << " " << '\n';
std::cout << x << " doubled is: " << doubleNumber(x) << '\n';

return 0;

}

Last edited 2 months ago by alb

0 Reply

alb

Reply to alb 21 January 24, 2024 5:11 am

To anyone reading this who knows better than I do:

How do I add a space in between the outputs in the program without having to add an entire line (std::cout << " " << '\n';) ?

Last edited 2 months ago by alb

0 Reply

Patrick

Reply to alb 22 January 31, 2024 11:52 am

std::cout << "This program will double the number inserted. " << std::endl << std::endl;
std::cout << std::endl << x << " doubled is: " << doubleNumber(x) << std::endl;

Also you do not need to define the doubleNumber function within the main function because it already registers it's presence by being contained within the same file above the main function.

0 Reply

Kai

Reply to alb 22 January 25, 2024 9:05 pm

For me doing endl + \n works. ill give an example

std :: cout << "Input a number: " << endl << "\n";

if you only have endl or \n then it will only go to the next line, but if you do both, then it will create the gap in between the terminal lines. I hope this is what you are looking for! (sorry if not)

0 Reply

Kai
Reply to Kai [23] January 25, 2024 9:09 pm

Just realized that you want a space in between your cin code aswell... I think this only works for cout
(im new, so not fully sure how it all works) Good luck :)

0 Reply

dylob
January 23, 2024 8:56 pm

Thank you so much for having this website; it's saving me a lot of money lol.

1 Reply

Chayim
January 21, 2024 10:29 pm

Why is it called argument when initializing a `var` with an outcome of a function? There’s no arguments here only taking the outcome value of function and initializing the `var`

Last edited 2 months ago by Chayim

0 Reply

Alex Author
Reply to Chayim [24] January 23, 2024 10:08 am

> Why is it called argument when initializing a var with an outcome of a function?

It isn't. Did I say it was somewhere?

> And same question to Why are values applied to a function argument called argument?

Not sure. The term comes from mathematics, but I don't know what the etymology of the term is.

0 Reply

Tortik
January 17, 2024 6:22 am

This is how I did it:

```
#include <iostream> // input/output library
#include <cstdlib> // pre-processor macro library

int returnDouble(int x /*user input*/) {
  return x*2;
}

int main() {
  //ask user to enter a number to use as an argument
  std::cout << "Please enter an integer: " << '\n';

  //list initialisation & variable assignment to pass as an argument
  int val = {};
  std::cin >> val;

  //call the returnDouble() function with defined argument and output its
return value
  std::cout << "The double of that vlaue is: " << returnDouble(val) << '\n';

  //indication that program has successfully executed
  return EXIT_SUCCESS;
}
```

Last edited 3 months ago by Tortik

2 Reply

Gripps
January 11, 2024 5:27 pm

#include <iostream>

/* my humble doubling machine! I quite like the brevity of the solution provided in the lesson, especially if the goal is to only create code that doubles a single integer.
I thought it would be nice to have the user's input as a separate function if I ended up needing it. So cool to see the different approaches to the same solution in C++!*/

int playerInput()
{
int input;
std::cout << "Enter an integer: ";
std::cin >> input;

return input;
}

int doubleNumber(int x)
{
return x * 2;
}

int main()
{
int keyboardInput{ playerInput() };
std::cout << keyboardInput << " doubled is " << doubleNumber(keyboardInput) << ". \n";

return 0;
}

Last edited 3 months ago by Gripps

2 Reply

Chayim
January 8, 2024 10:22 pm

You wrote:

However, what if we wanted to put the output line into its own function as well? You might try something like this:

```cpp
#include <iostream>

int getValueFromUser()
{
    std::cout << "Enter an integer: ";
    int input{};
    std::cin >> input;

    return input;
}

// This function won't compile
void printDouble()
{
    std::cout << num << " doubled is: " << num * 2 << '\n';
}

int main()
{
    int num { getValueFromUser() };

    printDouble();

    return 0;
}
```

But you can make it this way:

```
#include <iostream>

int getValueFromUser()
{
    std::cout << "Enter an integer: ";
    int input{};
    std::cin >> input;

    return input;
}

void printDouble()
{
int num { getValueFromUser() };
    std::cout << num << " doubled is: " << num * 2 << '\n';
}

int main()
{
printDouble();

    return 0;
}
```

Last edited 3 months ago by Chayim

0 Reply

Alex Author

Reply to Chayim 25 January 9, 2024 10:15 pm

Yes, but it doesn't make sense for `printDouble()` to ask the user for input. It's supposed to do printing, not input handling.

2 Reply

Fltyxy

January 6, 2024 2:17 am

#include <iostream>

int getValueFromUser() {
std::cout << "Enter ur integer number: ";
int input;
std::cin >> input;
return input;
}
void doubleNumber(int value) {
std::cout << "Your double number: " << value * 2;
}
int main() {

int num{ getValueFromUser() };
doubleNumber(num);
}

1 Reply

Links

1. https://www.learncpp.com/author/Alex/
2. https://www.learncpp.com/cpp-tutorial/overloading-the-increment-and-decrement-operators/
3. javascript:void(0)
4. https://www.learncpp.com/cpp-tutorial/introduction-to-local-scope/
5. https://www.learncpp.com/
6. https://www.learncpp.com/cpp-tutorial/void-functions-non-value-returning-functions/
7. https://www.learncpp.com/introduction-to-function-parameters-and-arguments/
8. https://www.learncpp.com/cpp-tutorial/introduction-to-iostream-cout-cin-and-endl/
9. https://gravatar.com/
10. https://www.learncpp.com/cpp-tutorial/introduction-to-function-parameters-and-arguments/#comment-595384
11. https://www.learncpp.com/cpp-tutorial/introduction-to-function-parameters-and-arguments/#comment-595234
12. https://www.learncpp.com/cpp-tutorial/introduction-to-function-parameters-and-arguments/#comment-594739
13. https://www.learncpp.com/cpp-tutorial/introduction-to-function-parameters-and-arguments/#comment-594634
14. https://www.learncpp.com/cpp-tutorial/introduction-to-function-parameters-and-arguments/#comment-594149
15. https://www.learncpp.com/cpp-tutorial/introduction-to-function-parameters-and-arguments/#comment-593883
16. https://www.learncpp.com/cpp-tutorial/introduction-to-function-parameters-and-arguments/#comment-593833
17. https://www.learncpp.com/cpp-tutorial/introduction-to-function-parameters-and-arguments/#comment-593733
18. https://www.learncpp.com/cpp-tutorial/introduction-to-function-parameters-and-arguments/#comment-593406
19. https://www.learncpp.com/cpp-tutorial/introduction-to-function-parameters-and-arguments/#comment-593029
20. https://www.learncpp.com/cpp-tutorial/introduction-to-function-parameters-and-arguments/#comment-592814

21. https://www.learncpp.com/cpp-tutorial/introduction-to-function-parameters-and-arguments/#comment-592804
22. https://www.learncpp.com/cpp-tutorial/introduction-to-function-parameters-and-arguments/#comment-592807
23. https://www.learncpp.com/cpp-tutorial/introduction-to-function-parameters-and-arguments/#comment-592897
24. https://www.learncpp.com/cpp-tutorial/introduction-to-function-parameters-and-arguments/#comment-592672
25. https://www.learncpp.com/cpp-tutorial/introduction-to-function-parameters-and-arguments/#comment-592018